Comparison of Various Encryption Techniques Based on Deterministic Chaos

Miroslav Popelka
Faculty of Applied Informatics
Tomas Bata University
Zlín, Czech Republic
e-mail: popelka@fai.utb.cz

***Abstract*—In this paper, six image encryption algorithms were considered in order to compare the influence of data shifting on encrypted data. Presented algorithms are based on a deterministic chaotic logistic map and shift the individual components of a pixel RGB (Red, Green, Blue) or complete color of the pixel to secure given input. The algorithms have been written in C# language and were adjusted to encrypt an image, nevertheless, they can be easily modified for any other multimedia file. Furthermore, two C# applications have been created. Chaos - Statistical testing application was created to evaluate histogram, sensitivity, correlation, entropy, and time consumption of the original and encrypted images. Additionally, another C# windows application was developed for the visualization and presentation of the generated chaotic data; furthermore, it provides basic encryption with various types of chaotic maps and dimensions.**

Keywords-Chaos; Chaotic Deterministic Map; Image Encryption; Pixels Shifting.

I. INTRODUCTION

Like many other great inventions, encryption was created in wartime in order to make a message impossible to read without some specific knowledge. Over time, many encryption techniques were developed to reach this goal with various mathematical and statistical knowledge. One of the interesting techniques for an image encryption in these days is chaotic encryption. In particular, discrete chaotic encryption is a widespread technique which uses a variety of deterministic map and differential equations.

Many technical papers were published on this topic. Discrete chaotic encryption algorithms for encryption of different types of multimedia files were published, for instance in [1], [2], and [3]. In these papers, multiple discrete chaotic maps were combined or chained in order to encrypt a required multimedia input file. For instance, L. Zhang et al. [1] present an image encryption algorithm based on "XOR plus mod" operation. The algorithm is designed to increase resistance in comparison with previously designed algorithms. Also, C. Li et al. [2] mention a chaotic image encryption algorithm using XOR operation; furthermore, they implement a circular bit shifting of image data. Total shuffling algorithm for image encryption is described in [3], where presented encryption algorithm combines two chaotic systems to improve the security level. To achieve this, the authors used a matrix in order to shift the position of image pixels. In [13], [14], and [15] the authors are using dispersion matrix to disperse data and provide more secure algorithms.

The purpose of all the mentioned algorithms was to improve the encryption process in terms of speed and security. In this paper, six algorithms with data shifting were developed.

Furthermore, two windows based applications for binary data sequence generation of the selected chaotic map and for analytical and statistical testing were created. The first application can encrypt or decrypt three types of multimedia files (image, text and binary file). Moreover, the encryption method can be selected from the list offered by the deterministic chaotic map; in addition, the initial condition parameters can be set up as well. The second application is used for the statistical evaluation of encrypted multimedia files in various tests (correlation, histograms, sensitivity, entropy, and time consumption).

In this paper, six image encryption methods using deterministic chaotic map are presented, especially a logistic map. These encryption algorithms can be easily transformed to encrypt any multimedia file.

The rest of the paper is structured as follows. Section II presents basic information about the logistic chaotic map. In Section II, we describe our proposed algorithms based on this logistic map. In Sections IV and V, two C# applications are presented. In Section VI, we evaluate and compare the data obtained in testing the proposed algorithms. Section VII provides a discussion of the results, and we conclude in Section VIII.

II. DETERMINISTIC CHAOTIC MAPS

The main property of chaotic dynamic systems is the sensitivity of initial conditions and control parameters and encryption algorithms benefit from this advantage. Discrete systems are mainly described by discrete formulas or differential equations, which represent their behavior in a short time period. In this paper, a basic deterministic logistic map is used.

A. Logistic map

This is one of the simple dynamical nonlinear systems, which shows chaotic behavior. A mathematical model of this map is described in (1) and a bifurcation diagram is shown in Figure 1.

$$x_{n+1} = r\, x_n\, (1-x_n). \quad (1)$$

 ISBN: 978-1-61208-493-0

Where ”xn” and “xn+1” are numbers between zero and one. “xn“ has an initial value usually set to 0.1. A parameter “r” is in the interval (0, 4]. The “r” parameter has a value equal to 1 at the start.

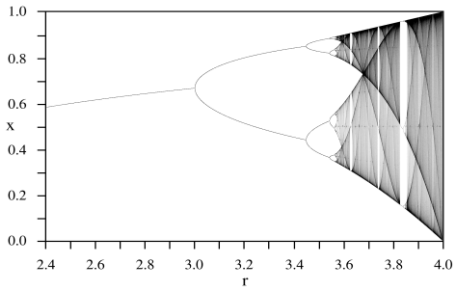

Figure 1. Bifurcation diagram of a logistic map [9]

III. ENCRYPTION APPLICATION

In this section, six encryption algorithms are described. Each algorithm is written in C# language and has this structure, which is described in Figure 2.

```
for (int i = 0; i < image.Height; i = i + 1)
{
   for (int j = 0; j < image.Width; j = j + 1)
   {
      original_pixel = image.GetPixel(i, j);
      byte R_component = (byte)(original_pixel.R ^ Ch_pole[ch_i]);
      byte G_component = (byte)(original_pixel.G ^ Ch_pole[ch_i + 1]);
      byte B_component = (byte)(original_pixel.B ^ Ch_pole[ch_i + 2]);
      byte A_component = (byte)(original_pixel.A ^ Ch_pole[ch_i + 3]);
      encrypted_pixel = Color.FromArgb(A_component, R_component,
     G_component, B_component);
      image.SetPixel(i, j, myRgbColor);
      ch_i = ch_i + 4;
```

Figure 2. A general structure of the encryption algorithms

The middle section is shown in Figure 2. It is different for all presented algorithms. This section is important for every algorithm which is described in the following sections.

A. Simple algorithm with XOR

The first algorithm performs only simple XOR operation with a binary data. This operation takes each pixel red, blue, green and alpha component and performs XOR operation with generated binary chaotic data. The individual value obtained is placed in the same pixel position where it was before. This algorithm can be seen in Figure 2.

B. Algorithm with a basic data shifting

The second presented algorithm described in Figure 3, is based on pixel shifting. The position of each pixel is given by original pixels position in X-th row and Y-th column. If pixel positions and generated chaotic data are known XOR operation is done for every single pixel. The foregoing implies that the final image will be reconstructed with the same pixels as in the original image, although their position will be changed.

```
new_row = (Ch_pole[ind] ^ lower_k) *
      (Ch_pole[ind + 1] ^ upper_k) % (image.Height);
new_col = (Ch_pole[ind + 2] ^ lower_l) *
      (Ch_pole[ind + 3] ^ upper_l) % (image.Width);

myRgbColor = image.GetPixel(new_col, new_row);

Encr_image.SetPixel(new_col, new_row, pixelColor);
Encr_image.SetPixel(l, k, myRgbColor);
```

Figure 3. An encryption algorithm with the basic data shifting

C. Algorithm with an advanced data shifting

In Figure 4 the third algorithm based on pixel shifting is described. In contrast with the previous shifting algorithm, where pixels are shifted, in this algorithm, positions of individual components (R, G, B, A) of all encrypted data are shifted. Exclusive disjunction of specific components and generated chaos is calculated and encrypted image is provided.

```
int new_row_red = (Ch_pole[ind] ) *
      (Ch_pole[ind + 3]) % (image.Height);
int new_col_red = (Ch_pole[ind]) *
      (Ch_pole[ind + 1]) % (image.Width);
int new_row_green = (Ch_pole[ind] ) *
      (Ch_pole[ind + 1] ) % (image.Height);
int new_col_green = (Ch_pole[ind] ) *
      (Ch_pole[ind + 2]) % (image.Width);
int new_row_blue = (Ch_pole[ind + 1] ) *
       (Ch_pole[ind + 3]) % (image.Height);
int new_col_blue = (Ch_pole[ind + 2] ) *
      (Ch_pole[ind + 3] ) % (image.Width);
int new_row_a = (Ch_pole[ind] ) *
       (Ch_pole[ind + 3]) % (image.Height);
int new_col_a = (Ch_pole[ind] ) *
      (Ch_pole[ind + 2] ) % (image.Width);
```

Figure 4. An encryption algorithm with a circle data shifting

D. Algorithm with a circle data shifting

The fourth tested algorithm, which can be seen in Figure 5, changes the position of pixels in circles with specific radius and angle. To deploy individual pixels in a circle, it is important to determine the radius and the angle from the generated chaotic data.

```
radius = ((int)angle * radius);
angle =( angle * ch_1 )% 360;

Point circle_point = new Point(0, 0);
circle_point.X = (int)(x + radius * Math.Cos(angle * (Math.PI / 180.0)));
circle_point.Y = (int)(y + radius * Math.Sin(angle * (Math.PI / 180.0)));
```

Figure 5. An encryption algorithm with a circle data shifting

 ISBN: 978-1-61208-493-0

Centers of the circles are located into a position of every pixel in the rectangular arrangement. After evaluating an angle and radius of the circle, it is necessary to define the new position and provide a completely encrypted image.

Due to the rectangular data arrangement in the original image, the algorithm had to deal with overflow. The overflow led to unwanted results. The solution how to compensate overflow can be seen in Figure 6.

```
if (Math.Abs(circle_point.X) >= image_height)
{
   int dif = ((Math.Abs(circle_point.X)) % image_height);
   if (dif == 0) { dif = 1; }
   if (ch_1 > 127)
   circle_point.X = dif;
   else
      circle_point.X = image_height-dif;
}
```

Figure 6. An overflow compensations for x coordinates

E. *Algorithm with a polynomial data shifting*

The fifth algorithm, displayed in Figure 7, also used the analytical mathematical function in order to change the position of pixels in the encrypted image. In this case, a polynomial function of a second degree was applied on the position of the individual pixel.

```
integers[0] = ch_pole[0] * x * x * x +
              ch_pole[1] * x * x +
              ch_pole[2] * x  +
              ch_pole[3] ;
integers[1] = ch_pole[3] * y * y * y +
              ch_pole[2] * y * y +
              ch_pole[1] * y  +
              ch_pole[0] ;

Point new_spot= new Point(0,0);

new_spot.X = (Math.Abs(integers[0])) % image_height;
new_spot.Y = (Math.Abs(integers[1] ))% image_width;
```

Figure 7. An encryption algorithm with a polynomial data shifting

In this algorithm, an overflow needs to be reckoned. The solution is very similar to the previous one, although the overflow compensation is slightly more often applied due to polynomial function characteristic. The overflow compensation can be seen on the last two lines in Figure 7.

F. *Algorithm with change color/position*

Finally, this encryption algorithm combines a position shifting with pixel R, G, B and A components of the color, in order to change the position of given pixel. This algorithm can be seen in Figure 8.

The first step was to decompose pixel colors into R, G, B and A components.

Two components and a number of rows (X) are XORed and used for determination of the new X position.

Two other parts of the pixel color are processed in the same way to obtain a new column number (Y).

```
byte red = (byte)(pixelColor.R ^ Ch_pole[ind]);
byte green = (byte)(pixelColor.G ^ Ch_pole[ind + 1]);
byte blue = (byte)(pixelColor.B ^ Ch_pole[ind + 2]);
byte alpha = (byte)(pixelColor.A ^ Ch_pole[ind + 3]);

pixelColor = Color.FromArgb(alpha, red, green, blue);
myRgbColor = image.GetPixel(new_col, new_row);

byte red_n = (byte)(myRgbColor.R ^ Ch_pole[ind+3]);
byte green_n = (byte)(myRgbColor.G ^ Ch_pole[ind + 2]);
byte blue_n = (byte)(myRgbColor.B ^ Ch_pole[ind + 1]);
byte alpha_n = (byte)(myRgbColor.A ^ Ch_pole[ind]);

myRgbColor = Color.FromArgb(alpha_n, red_n, green_n, blue_n);
```

Figure 8. An overflow compensations for x coordinates

IV. APPLICATION DESCRIPTION

The main window of this application was designed to generate chaotic pseudo-random data. This data were used together with the original data to perform an encryption. The main window of the application is divided into three main control parts.

A. *Settings section*

The most important part of the main window is the settings section, which is situated on the very left side of the main window. Selectors and two text boxes can also be found. Selectors are designated to select the dimension of the chaotic map and for selection of the specific deterministic chaotic map in the second selector. According to the previous selections, texts in text boxes below are dynamically updated and they contain parameters and initial conditions according to the selected dimension and type of deterministic chaotic map. Part of this setting section can be seen in Figure 9.

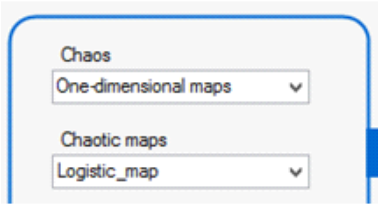

Figure 9. Selection of dimension and type of chaotic map

B. *Control section*

The application control elements are placed in the middle between settings and the results section. This middle control section contains buttons to change the content of the results section. There are a decryption, an encryption and also a

visualization data buttons. Figure 10 shows the main control buttons in the application.

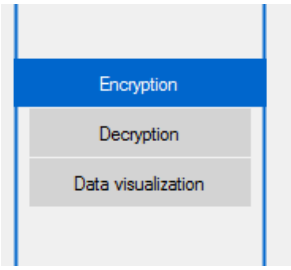

Figure 10. Control buttons

C. *Results section*

In terms of displaying the results, this section is the most important of all. In the result section, decrypted files, images, and text files can be encrypted. The results obtained are displayed in appropriate form for observation. An example of the obtained result is shown in Figure 11.

Figure 11. Example of application results section

Naturally, this application does not serve only for the generation of chaotic data. In addition, it can be used for encryption of different multimedia files. It is necessary to attach a list of parameters and initial conditions to the encrypted file. These items are needed for successful decryption.

Of course, the parameters and the initial conditions cannot be transported non-encrypted. On this attached information is applied well-known RSA encryption algorithm, which works with the public and private key.

V. TESTING APPLICATION

Analytical and statistical tests are an integral part of the development of encryption algorithm. For these reasons, the testing application was created in order to complete all mentioned tests.

In this test application, it is possible to test input non-encrypted and encrypted file in various tests such as histogram, image entropy, sensitivity and correlation of every pixel. The Chaos - Statistical testing application can be seen in Figure.12.

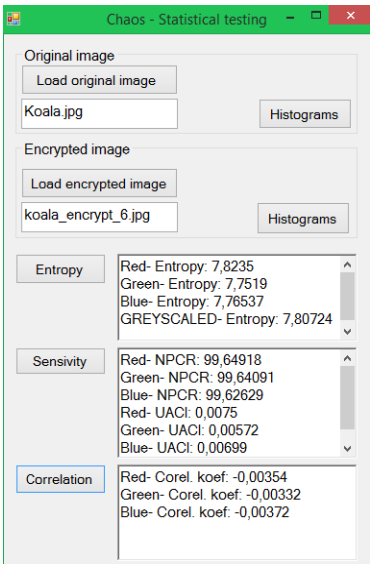

Figure 12. The Chaos - Statistical testing

Moreover, the disturbance of specific color in an original and in an encrypted image can be estimated in this application. Time consumption of each presented algorithms can be also measured by the application. The encryption time depends on an image size, although images must have the same resolution, otherwise, the result will not be possible to compare. Furthermore, comparison of a color and a grayscale image is not relevant and the given result will not be definitive.

The technique of time measurement will start at the beginning of the encryption algorithm itself, after producing of the chaotic data. All measurements were performed on the same computer with the same hardware equipment to prevent a hardware performance error.

VI. RESULTS

The input data for all six methods were an identical collection of images. This image collection covers both a grayscale and color images with various resolutions ranging from 50 x 50 pixels to 1024 x 768 pixels. The chosen resolution range is sufficient to show dependency on the number of pixels.

The image collection was divided into three equal parts by the image resolution. Every collection part contains ten color and ten grayscale images with the corresponding resolution. The comparison of obtained results from an individual algorithm can be seen in following Figure 13. and tables (Table I – Table VI).

 ISBN: 978-1-61208-493-0

A. *Time consumption comparisons*

Figure 13. represents time consumptions of every presented encryption algorithm in the selected image collection.

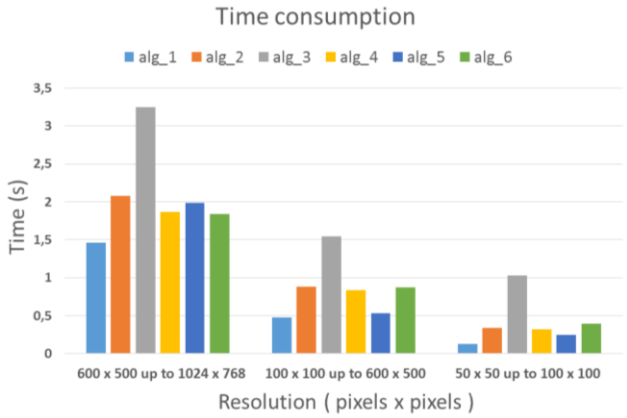

Figure 13. The average time consumption for each algorithm

B. *Statistical comparisons*

The statistical results were estimated for the color and grayscale images within the specific collection and these results are averaged and displayed in the tables (Table I – Table VI). Individual statistical test and their formulas are described in this section.

1) *Image entropy*

It denotes the probability of the single pixel color in the encrypted image (2).

The ideal image entropy for one color image is 8. In this paper, the individual color components (R, G, B) are averaged into a single value in order to compare with results obtained from grayscale images.

$$H(s) = -\sum_{i=0}^{2^n-1} P(s_i)\log_2[P(s_i)]. \quad (2)$$

2) *Sensitivity analysis*

To test the influence of the change of image pixel on the encrypted image, two statistical coefficients are often used. The first coefficient is net pixel change rate (NPCR) (3).

$$NPCR = \frac{\sum_{ij} D(i,j)}{M \times N} \times 100\% . \quad (3)$$

and the second is the unified average changing rate (UACI) (4).

$$\mathrm{UACI} = \frac{1}{\mathrm{M} \times \mathrm{N}} \cdot \sum_{\mathrm{ij}} \frac{|\mathrm{c}_1(\mathrm{i},\mathrm{j}) - \mathrm{c}_2(\mathrm{i},\mathrm{j})|}{255} \times 100\,\% \quad (4)$$

These two coefficients state for an encrypted image pixel sensitivity on the original image pixels.

3) *Correlation*

The correlation is the dependency between a pixel in the original and encrypted image. The ideal value is 0. The correlation can be calculated by (5).

$$\gamma_{xy} = \frac{Conv(x,y)}{\sqrt{D(x)} \cdot \sqrt{D(y)}} . \quad (5)$$

4) *Measured and averaged statistical results*

This section displays Table I – Table VI in order to compare individual results in the individual statistical tests.

TABLE I. RESULTS FOR GRAYSCALE IMAGES WITH RESOLUTION (600 X 500) UP TO (1024 X 768) PIXELS

Algorithm No.	Grayscale			
	Entropy	*NPCR*	*UACI*	*Correlation*
1	7.8472	99.4948	0.0089	-0.0003
2	7.7072	99.5967	0.0285	0.0077
3	7.7861	99.6149	0.0045	0.0144
4	7.5999	99.6351	0.0065	-0.0010
5	7.6098	99.6063	0.0052	-0.0052
6	7.9132	99.7198	0.0061	-0.0035

TABLE II. STATISTICAL RESULTS FOR COLOR IMAGES WITH RESOLUTION (600 X 500) UP TO (1024 X 768)PIXELS

Algorithm No.	Color			
	Entropy	*NPCR*	*UACI*	*Correlation*
1	7.7862	99.4968	0.0085	-0.0002
2	7.7339	99.5899	0.0254	0.0058
3	7.7874	99.6086	0.0040	0.0134
4	7.6011	99.6337	0.0061	-0.0021
5	7.5998	99.5979	0.0058	-0.0061
6	7.9298	99.7301	0.0065	-0.0039

TABLE III. STATISTICAL RESULTS FOR GRAYSCALE IMAGES WITH RESOLUTION (100 X 100) UP TO (600 X 500) PIXELS

Algorithm No.	Grayscale			
	Entropy	*NPCR*	*UACI*	*Correlation*
1	7.7482	99.5238	0.0085	-0.0005
2	7.6972	99.5836	0.0294	0.0076
3	7.8001	99.6259	0.0037	0.0140
4	7.5874	99.6201	0.0073	-0.0009
5	7.7001	99.6086	0.0043	-0.0048
6	7.8513	99.7205	0.0055	-0.0033

 ISBN: 978-1-61208-493-0

TABLE IV. STATISTICAL RESULTS FOR COLOR IMAGES WITH RESOLUTION (100 X 100) UP TO (600 X 500) PIXELS

Algorithm No.	Color			
	Entropy	*NPCR*	*UACI*	*Correlation*
1	7.7663	99.5337	0.0071	-0.0004
2	7.6881	99.5741	0.0282	0.0088
3	7.8555	99.5993	0.0039	0.0190
4	7.6577	99.6009	0.0066	-0.0014
5	7.7305	99.6255	0.0050	-0.0050
6	7.9012	99.7318	0.0056	-0.0048

TABLE V. STATISTICAL RESULTS FOR GRAYSCALE IMAGES WITH RESOLUTION (50 X 50) UP TO (100 X 100) PIXELS

Algorithm No.	Grayscale			
	Entropy	*NPCR*	*UACI*	*Correlation*
1	7.7919	99.4819	0.0080	-0.0010
2	7.7872	99.5954	0.0304	0.0084
3	7.7189	99.5949	0.0063	0.0159
4	7.5867	99.6256	0.0085	-0.0008
5	7.6672	99.6163	0.0054	-0.0059
6	7.9624	99.7298	0.0070	-0.0031

TABLE VI. STATISTICAL RESULTS FOR COLOR IMAGES WITH RESOLUTION (50 X 50) UP TO (100 X 100) PIXELS

Algorithm No.	Color			
	Entropy	*NPCR*	*UACI*	*Correlation*
1	7.8211	99.4997	0.0075	-0.0006
2	7.7344	99.5954	0.0259	0.0083
3	7.7299	99.5925	0.0056	0.0136
4	7.5616	99.6311	0.0093	-0.0006
5	7.7493	99.6327	0.0033	-0.0070
6	7.9338	99.7213	0.0064	-0.0039

VII. DISCUSSION

In this paper, six shifting chaotic encryption techniques have been presented. Every algorithm was tested under various tests, such as correlation, image sensitivity, and time consumption. All algorithms are based on the pixel shifting and XOR operation with the chaotic data. The results obtained from six image encryption algorithms were compared and the differences between these results were negligible. All six algorithms provide the same level of security. This fact is supported by Tables I – VI. Especially, correlation coefficients display the most accurate information about the individual pixel similarity of the original and encrypted image. Nevertheless, an entropy in the last presented algorithm is about 7.9, which means, how many pixels have a random color. In other words, how much from the encrypted image is similar to the original. As can be seen from presented results these techniques provided adequate results according to their complexity. The results from statistical tests and the time consumption of an individual test were not as expected. Especially, results of entropy, NPCR and UACI did not provide such results in comparison with [1], [2] and [3]. However, the last presented algorithm showed that a position shifting combined together with the color components can produce satisfactory results in a reasonable time. This algorithm and its variation will be examined in future research.

VIII. CONCLUSION

The main aim of this work was to compare and evaluate position shifting algorithms. Six different image encryption algorithms based on the chaotic discrete logistic map were created. Each algorithm was tested in various tests (correlation, histograms, sensitivity, image entropy, and time consumption). These tests were performed on the collection of the test images with a resolution of 50 x 50 pixels up to 1024 x 786 pixels and results were shown in tables for every presented algorithm. In addition, the time consumption has been measured for every algorithm. The last presented algorithm has most significant results from all the created algorithms.

ACKNOWLEDGMENT

This work was supported by Internal Grant Agency of Tomas Bata University under the project No. IGA/FAI/2016/024.

REFERENCES

[1] L. Zhang, X. Liao and X. Wang, “An image encryption approach based on chaotic maps” Chaos, Solitons & Fractals, May 2005, pp. 759-765, doi: 10.1016/j.chaos.2004.09.035.

[2] C. Li, S. Li, G. Alvarez, G. Chen and K. T. Lo,“Cryptanalysis of a chaotic encryption system” Physics Letters A, Sept. 2007, pp. 23-30, doi: 10.1016/j.physleta.2007.04.023.

[3] T. Gao and Z. Chen,“Image encryption based on a new total shuffling algorithm” Chaos, solitons & fractals, Oct. 2008, pp. 213–220, doi: 10.1016/j.chaos.2006.11.009.

[4] S. Fu-Yan, L. Shu-Tang and L. Zong-Wang “Image encryption using high-dimension chaotic system” Chinese Physics, Dec. 2007, pp. 3616 - 3624, doi: 10.1088/1009-1963/16/12/011

[5] C. Y. Chee and D. Xu, “Chaotic encryption using discrete-time synchronous chaos” Physics Letters A, Dec. 2006, pp. 284-292, doi: 10.1016/j.physleta.2005.08.082

[6] Zhang, W., Wong, K. W., Yu, H., & Zhu, Z. L. (2013). An image encryption scheme using reverse 2-dimensional chaotic map and dependent diffusion. Communications in Nonlinear Science and Numerical Simulation, 18(8), 2066-2080.

[7] S. El Assad, M. Farajallah, “A new chaos-based image encryption system”, Signal Processing: Image Communication, vol. 41, 2016, pp. 144-157.

[8] M. Farajallah, S. El Assad, O. Deforges, “Fast and secure chaos-based cryptosystem for images”, International Journal of Bifurcation and Chaos, IJBC, February 2016, Vol. 26, No. 02, pp. 1650021-1 1650021-21. DOI: 10.1142/S0218127416500218.

[9] Zhang, X., Zhao, Z., & Wang, J. (2014). Chaotic image encryption based on circular substitution box and key stream buffer. Signal Processing: Image Communication, 29(8), 902-913.

 ISBN: 978-1-61208-493-0